Bilkent University

Department of Computer Engineering

CS353 – Database Systems

Project Final Report

Airline Company Data Management System

Website: http://cs353group20.github.io/

Group 20:

- İrem Ergün
- Nihat Eren Ekinci
- Ömer Eren
- Turan Kaan Elgin

Table of Contents

1. Project Description

The project is an airline company data management system which is used by both the employees and the customers of the airline company. Access control is done via the login phase of our system. All the users first have to sign up for the system and then they can login to do what they have been signed up for.

The administrator of the system decides the role of each user who has signed up. Also, he can add, remove or alter every entity in the system such as adding flights, delaying flights, assigning flight personnel, and so on.

Essentially, the customers can buy or reserve tickets via the system. Customers can view all the flights and search for a destination to travel. The system shows both the direct flight and connecting flight options from a specific location to another. After choosing their flight option, the customers can proceed to either buying or reserving the ticket. They also have the option to cancel a reservation or a purchased ticket. However, they are faced with a fee if they cancel their ticket after a specific deadline. Based on their purchases, they will receive some promotions (flight/food/store) given by the system. Moreover, the customers are given the ability to browse stores of a specific airport.

Crew of the flights (pilots and flight attendants) can view their flight schedule via the system. Pilots can also view the information about the planes that they will fly. In addition, flight attendants are responsible for recording the usage of food promotion of a customer during a flight using the system.

Ticket staff is responsible for selling tickets and recording usage of flight promotions of the customers. The extra luggage information of a customer is entered by ticket staff using the system during the ticket sale. The customers are then faced with an additional fee if they want to bring extra luggage to the flight.

Store staff is responsible for entering the usage of store promotions of a customer via our system.

In addition, flight histories of all users are preserved in the system in case the users want to view their flights later on.

2. E/R Model

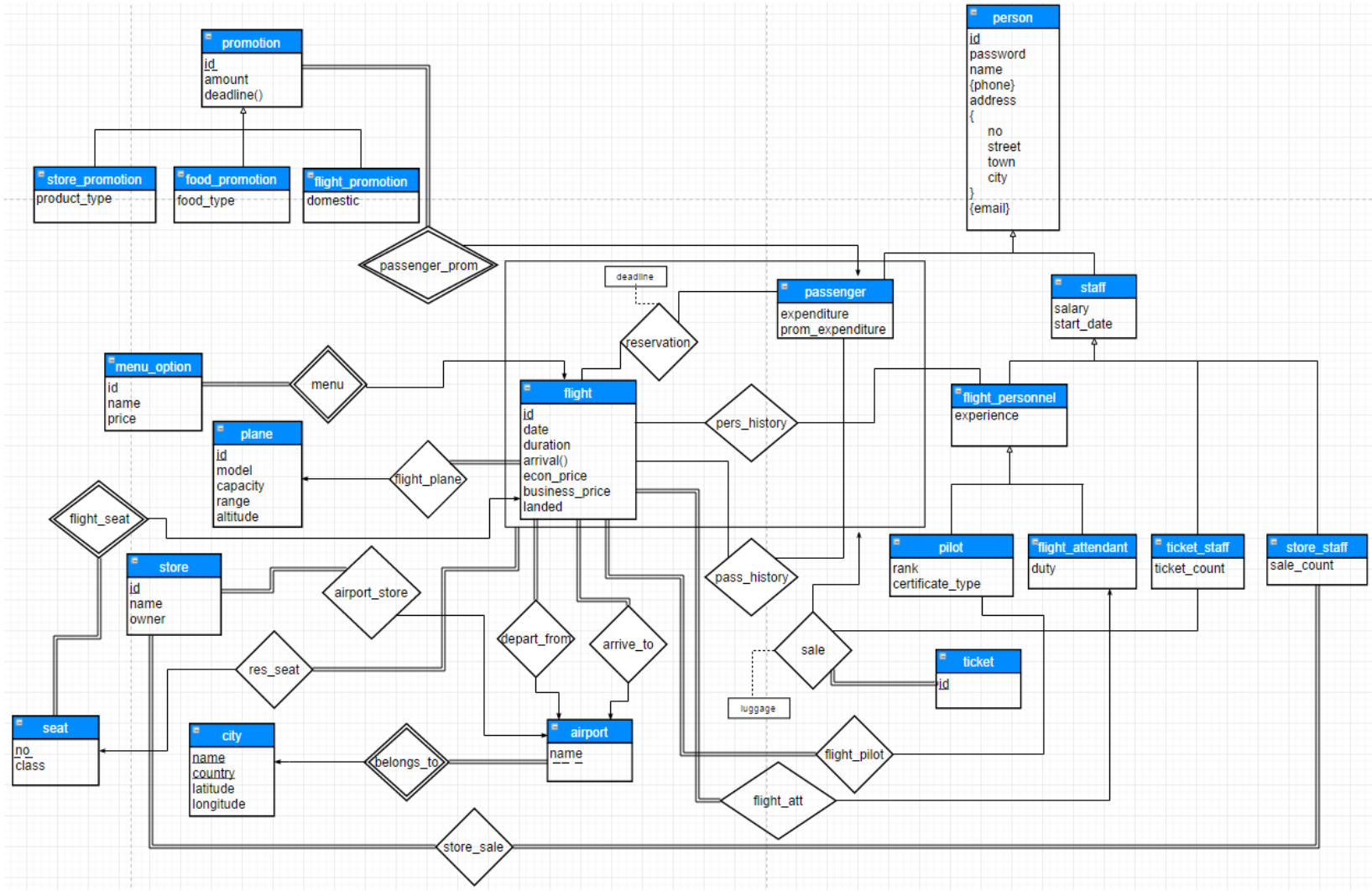


Figure 1.1 – E/R Diagram

3. Table Schemas

3.1 Person

Relational Model:
person(person_id, password, person_name, address_no, street, town, city)

Nontrivial Functional Dependencies:
person_id -> password person_name address_no street town city

Candidate Keys:
{(person_id)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE person (
        person_id int PRIMARY KEY AUTO_INCREMENT,
        password varchar(40) NOT NULL,
        person_name varchar(40) NOT NULL,
        address_no int NOT NULL,
        street varchar(40) NOT NULL,
        town varchar(40) NOT NULL);
```

3.2 PersonPhone

Relational Model:
person_phone(<u>person_id, phone</u>)
person_id: FK to person

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(person_id, phone)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE person_phone (
    person_id int,
    phone varchar(20),
    PRIMARY KEY(person_id, phone),
    FOREIGN KEY(person_id) referencing person);
```

3.3 PersonEmail

Relational Model:
person_email(<u>person_id, email</u>)
person_id: FK to person

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(person_id, email)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE person_email (
    person_id int,
    email varchar(40),
    PRIMARY KEY(person_id, email),
    FOREIGN KEY(person_id) referencing person);
```

3.4 City

Relational Model:
city(city_name, country, latitude, longitude)

Nontrivial Functional Dependencies:
city_name country -> latitude longitude

Candidate Keys:
{(city_name, country)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE city (
        city_name varchar(40),
        country varchar(40),
        latitude numeric(8,5) NOT NULL,
        longitude numeric(8,5) NOT NULL,
        PRIMARY KEY(city_name, country),
        check(latitude >= 0 and latitude < 360 and
                longitude >= 0 and longitude < 180));
```

3.5 Airport

Relational Model:
airport(<u>airport_name, city_name, country</u>)
(city_name, country): FK to city

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(airport_name, city_name, country)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE airport(
        airport_name varchar(40),
        city_name varchar(40),
        country varchar(40),
        PRIMARY KEY(airport_name, city_name, country),
        FOREIGN KEY (city_name, country) references city);
```

3.6 Store

Relational Model:
store(store_id, store_name, owner, airport_name, city_name, country)
(airport_name, city_name, country): FK to airport

Nontrivial Functional Dependencies:
store_id -> store_name, owner, airport_name, city_name, country

Candidate Keys:
{(store_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE store (
    store_id int PRIMARY KEY AUTO_INCREMENT,
    store_name varchar(40) NOT NULL,
    owner  varchar(40),
    airport_name varchar(40),
    city_name varchar(40),
    country varchar(40),
    FOREIGN KEY (airport_name, city_name, country) references airport);
```

3.7 Passenger

Relational Model:
passenger(pass_id, expenditure, prom_expenditure)
pass_id: FK to person(person_id)

Nontrivial Functional Dependencies:
pass_id -> expenditure prom_expenditure

Candidate Keys:
{(pass_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE passenger (
        pass_id int PRIMARY KEY,
        expenditure numeric(4,2) NOT NULL,
        prom_expenditure numeric(4,2) NOT NULL,
        FOREIGN KEY(pass_id) references person(person_id));
```

3.8 Staff

Relational Model:
staff(staff_id, salary)
staff_id: FK to person(person_id)

Nontrivial Functional Dependencies:
staff_id -> salary

Candidate Keys:
{(staff_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE staff (
        staff_id int PRIMARY KEY,
        salary   numeric(12,2) NOT NULL,
        FOREIGN KEY(staff_id) references person(person_id));
```

3.9 FlightPersonnel

Relational Model:
flight_personnel(flight_pers_id, experience)
flight_pers_id: FK to staff(staff_id)

Nontrivial Functional Dependencies:
flight_pers_id -> experience

Candidate Keys:
{(flight_pers_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_personnel (
    flight_pers_id int PRIMARY KEY,
    experience int NOT NULL,
    FOREIGN KEY(flight_pers_id) references staff(staff_id));
```

3.10 Pilot

Relational Model
pilot(pilot_id, rank, certificate_type)
pilot_id: FK to flight_personnel(flight_pers_id)

Nontrivial Functional Dependencies
pilot_id -> rank certificate_type

Candidate Keys
{(pilot_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE pilot (
    pilot_id int PRIMARY KEY,
    rank int NOT NULL,
    certificate_type enum('sport', 'recreational', 'private', 'commercial', 'instructor', 'airline
            transport') NOT NULL,
    FOREIGN KEY(pilot_id) references flight_personnel(flight_pers_id));
```

3.11 FlightAttendant

Relational Model
flight_attendant(att_id, duty)
att_id: FK to flight_personnel(flight_pers_id)

Nontrivial Functional Dependencies
att_id -> duty

Candidate Keys
{(att_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE flight_attendant (
    att_id int PRIMARY KEY,
    duty varchar(40) NOT NULL,
    FOREIGN KEY(att_id) references flight_personnel(flight_pers_id));
```

3.12 TicketStaff

Relational Model
ticket_staff(ticket_staff_id, ticket_count)
ticket_staff_id: FK to staff(staff_id)

Nontrivial Functional Dependencies
ticket_staff_id ->ticket_count

Candidate Keys
{(ticket_staff_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE ticket_staff (
    ticket_staff_id int PRIMARY KEY,
    ticket_count int NOT NULL,
    FOREIGN KEY(ticket_staff_id) references staff(staff_id));
```

3.13 StoreStaff

Relational Model

store_staff(store_staff_id, sale_count, store_id)

store_staff_id: FK to staff(staff_id)

store-id: FK to store

Nontrivial Functional Dependencies

store_staff_id ->sale_count store_id

Candidate Keys

{(store_staff_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE store_staff (
        store_staff_id int PRIMARY KEY,
        sale_count int NOT NULL,
        store_id int NOT NULL,
        FOREIGN KEY(store_staff_id) references staff(staff_id),
        FOREIGN KEY(store_id) references store);
```

3.14 Promotion

Relational Model
promotion(pass_id, prom_id, amount)
pass_id: FK to passenger

Nontrivial Functional Dependencies
pass_id prom_id -> amount deadline

Candidate Keys
{(pass_id, prom_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE promotion (
    pass_id int,
    prom_id int AUTO_INCREMENT,
    amount int NOT NULL,
    PRIMARY_KEY(pass_id, prom_id),
    FOREIGN KEY(pass_id) referencing passenger);
```

3.15 PromotionDeadline

Relational Model:
promotion_deadline(amount, deadline)
amount: FK to promotion

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(amount, deadline)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE promotion_deadline(
    amount int,
    deadline date,
    PRIMARY KEY(amount, deadline),
    FOREIGN KEY(amount) referencing promotion);
```

3.16 StorePromotion

Relational Model

store_promotion(pass_id, prom_id, product_type)

pass_id: FK to passenger

prom_id: FK to promotion

Nontrivial Functional Dependencies

pass_id prom_id -> product_type

Candidate Keys

{(pass_id, prom_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE store_promotion (
        pass_id int,
        prom_id int,
        product_type enum('alcohol', 'normal') NOT NULL,
        PRIMARY_KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger
        FOREIGN KEY(prom_id) referencing promotion);
```

3.17 FoodPromotion

Relational Model

food_promotion(pass_id, prom_id, food_type)

pass_id: FK to passenger

prom_id: FK to promotion

Nontrivial Functional Dependencies

pass_id prom_id -> food_type

Candidate Keys

{(pass_id, prom_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE food_promotion (
        pass_id int,
        prom_id int,
        food_type enum('meal', 'drink') NOT NULL,
        PRIMARY_KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger
        FOREIGN KEY(prom_id) referencing promotion);
```

3.18 FlightPromotion

Relational Model:
flight_promotion(pass_id, prom_id, domestic)
pass_id: FK to passenger
prom_id: FK to promotion

Nontrivial Functional Dependencies:
pass_id, prom_id -> domestic

Candidate Keys:
{(pass_id, prom_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_promotion (
        pass_id int,
        prom_id int,
        domestic binary NOT NULL,
        PRIMARY KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger
        FOREIGN KEY(prom_id) referencing promotion);
```

3.19 Plane

Relational Model:
plane(plane_id, model)

Functional Dependencies:
plane_id -> model

Candidate Keys:
{(plane_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE plane(
        plane_id int PRIMARY KEY,
        model varchar(20) NOT NULL,
        FOREIGN KEY (model) references plane_model);
```

3.20 PlaneModel

Relational Model:
plane_model(model, capacity, range, altitude)
model: FK to plane

Functional Dependencies:
model -> capacity range altitude

Candidate Keys:
{(model)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE plane_model(
    model varchar(20) PRIMARY KEY,
    capacity int NOT NULL,
    range numeric(5,2) NOT NULL,
    altitude numeric(5,2) NOT NULL,
    FOREIGN KEY(model) referencing plane);
```

3.21 Flight

Relational Model:
flight(flight_id, date, plane_id, dep_airport_id, arr_airport_id, duration, econ_price, business_price)
plane_id: FK to plane
dep_airport_id: FK to airport(airport_id)
arr_airport_id: FK to airport(airport_id)

Functional Dependencies:
flight_id -> date plane_id dep_airport_id arr_airport_id duration econ_price business_price

Candidate Keys:
{(flight_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight (
    flight_id int PRIMARY KEY AUTO_INCREMENT,
    date DATE NOT NULL,
    plane_id int,
    dep_airport_name varchar(40),
    dep_city_name varchar(40),
    dep_country varchar(40),
    arr_airport_ name varchar(40),
    arr_city_name varchar(40),
    arr_country varchar(40),
    duration numeric(3,2) NOT NULL,
    econ_price numeric(6,2),
    business_price numeric(6,2),
    FOREIGN KEY (plane_id) references plane,
    FOREIGN KEY(dep_airport_id) references airport(airport_id),
    FOREIGN KEY(arr_airport_id) references airport(airport_id));
```

3.22 FlightArrival

Relational Model:
flight_arrival(<u>date, duration, arrival</u>)
(date, duration): FK to flight

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(date, duration, arrival)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_arrival (
    date DATE,
    duration numeric(3,2),
    arrival DATE,
    PRIMARY KEY(date, duration, arrival),
    FOREIGN KEY(date, duration) referencing flight);
```

3.23 Seat

Relational Model:
seat(flight_id, no, class)
flight_id: FK to flight

Functional Dependencies:
flight_id no -> class

Candidate Keys:
{(flight_id, no)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE seat (
    flight_id int,
    no int,
    class enum('econ', 'business') NOT NULL,
    PRIMARY KEY (flight_id, no),
    FOREIGN KEY (plane_id) references plane);
```

3.24 MenuOption

Relational Model:
menu_option(flight_id, option_id, option_name, price)
flight_id: FK to flight

Functional Dependencies:
flight_id option_id -> option_name price

Candidate Keys:
{(flight_id, option_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE menu_option(
        flight_id int,
        option_id int,
        option_name varchar(40) NOT NULL,
        PRIMARY KEY (flight_id, option_id),
        FOREIGN KEY (flight_id) references flight);
```

3.25 Reservation

Relational Model:
reservation(flight_id, pass_id, deadline, seat_no)
pass_id: FK to passenger
(flight_id, seat_no): FK to seat

Functional Dependencies:
(flight_id, pass_id) -> deadline seat_no

Candidate Keys:
{(flight_id, pass_id)}

Normal Form:
```
CREATE TABLE reservation(
        flight_id int,
        pass_id int,
        deadline date NOT NULL,
        seat_no int NOT NULL,
        PRIMARY KEY(flight_id, pass_id),
        FOREIGN KEY(pass_id) references passenger
        FOREIGN KEY(flight_id, seat_no) references seat);
```

3.26 PassengerHistory

Relational Model:
pass_history(flight_id, pass_id)
flight_id: FK to flight
pass_id: FK to passenger

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, pass_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE pass_history (
    flight_id int,
    pass_id int,
    PRIMARY KEY(flight_id, pass_id),
    FOREIGN KEY(flight_id) references flight,
    FOREIGN KEY(pass_id) references passenger);
```

3.27 PersonnelHistory

Relational Model:
pers_history(flight_id, flight_pers_id)
flight_id: FK to flight
flight_pers_id: FK to flight_personnel

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, flight_pers_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE pers_history(
    flight_id int,
    pers_id int,
    PRIMARY KEY(flight_id, flight_pers_id),
    FOREIGN KEY(flight_id) references flight,
    FOREIGN KEY(flight_pers_id) references flight_personnel);
```

3.28 Flight-Pilot Relationship

Relational Model:
flight_pilot(flight_id, pilot_id)
flight_id: FK to flight
pilot_id: FK to pilot

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, pilot_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_pilot (
        flight_id int,
        pilot_id int,
        PRIMARY KEY(flight_id, pilot_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(pilot_id) references pilot);
```

3.29 Flight – FlightAttendant Relationship

Relational Model:
flight_att(flight_id, att_id)
flight_id: FK to flight
att_id: FK to flight_attendant

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, att_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_att(
    flight_id int,
    att_id int,
    PRIMARY KEY(flight_id, att_id),
    FOREIGN KEY(flight_id) references flight,
    FOREIGN KEY(att_id) references flight_attendant);
```

3.30 Ticket

Relational Model:
ticket(ticket_id, flight_id, pass_id, staff_id, luggage)

Nontrivial Functional Dependencies:
ticket_id -> price flight_id pass_id staff_id luggage

Candidate Keys:
{(ticket_id)}

Table Definition:

```
CREATE TABLE ticket(
        ticket_id int,
        flight_id int NOT NULL,
        pass_id int NOT NULL,
        staff_id int,
        luggage int NOT NULL,
        PRIMARY KEY(ticket_id),
        FOREIGN KEY(flight_id, pass_id) references reservation,
        FOREIGN KEY(staff_id) references ticket_staff);
```

4 Implementation Details

We implemented the project by using Python language for application logic and MySQL for database queries. Also for implementing the web interface, HTML, CSS and JavaScript is used. Integrating the interface to the application logic is done with a micro framework of Python, called Flask. It is a useful framework such that it does not require any particular library for implementing the system, instead it can be integrated to any database system.

By implementing the project in Python, we were able to use modularity and especially the Model-View-Controller architectural style. MVC style is accomplished via seperating the database layer, application logic layer, and user interface layer.

Database layer is implemented by using two Pyhton modules which do not include Flask. One of this modules is for instantiating the database features such as tables, views and triggers. The other module is used for defining functions for each query in the system. By making this separation, application logic layer does not use SQL features, it just calls the functions abstracted in the module. The modules connect to the database server by using the particular library of Python. So, this layer is the model of the system.

Application logic layer is implemented by using Flask framework. It can connect to HTML and CSS files and accomplish form validation. So, this layer provides the connection between web interface and SQL queries and data, as the middle layer. It represents the controller of the system.

In the user interface layer, HTML, CSS and JavaScript are used and they are seperated from the other layers.

The difficulty encountered in the implementation is about MySQL such that it does not support some properties of SQL. These properties are “except” and “with” keywords and also assertions. So, different ways are used for implementing those.

5 Advanced DB Features

5.1 Secondary Indices

- Since users will login to the system via their emails, email attribute of person table will be searched frequently.

```
CREATE INDEX email_index USING BTREE ON person(email);
```

- Since users will search flights by their source and destination cities, they will be searched frequently.

```
CREATE INDEX source_city_index USING BTREE ON flight(dep_city_name);

CREATE INDEX dest_city_index USING BTREE ON flight(arr_city_name);
```

- Since planes are assigned to fligths by considering their ranges, the range attribute of plane table will be searched frequently.

```
CREATE INDEX plane_range_index USING BTREE ON plane(range);
```

5.2 Advanced Features

- For displaying flight histories of passengers and flight personnel, two views are created as the following.

```
cursor.execute( “CREATE VIEW pass_history_view AS SELECT * FROM pass_history WHERE pass_id = @pass_id”)

cursor.execute( “CREATE VIEW pers_history_view AS SELECT * FROM pers_history WHERE flight_pers_id = @flight_pers_id”)
```

@pass_id and @flight_pers_id are the ID numbers of the users currently signed in. When displaying their histories, the view are used as the following.

```
cursor.execute( “SELECT * FROM pass_history_view”)

cursor.execute( “SELECT * FROM pers_history_view”)
```

- When a flight is landed, it should be removed from the schedules of flight personnel. For this purpose, the following triggers are used.

```
cursor.execute( “CREATE TRIGGER land_flight_att AFTER UPDATE ON flight FOR EACH ROW
DELETE FROM flight_att WHERE flight_id = new.flight_id and new.landed = 1”)

cursor.execute( “CREATE TRIGGER land_flight_pilot AFTER UPDATE ON flight FOR EACH ROW
DELETE FROM flight_pilot WHERE flight_id = new.flight_id and new.landed = 1”)
```

5.3 Reports

- A report is used for displaying the promotions of each passenger.

Query:

```
SELECT pass_id, person_name, price, type AS CASE
                        WHEN prom_id IN (SELECT prom_id
                                         FROM food_promotion) THEN “Food”
                        WHEN prom_id IN (SELECT prom_id
                                         FROM store_promotion) THEN “Store”
                        WHEN prom_id IN (SELECT prom_id
                                         FROM flight_promotion) THEN “Flight”
                        END
FROM (person JOIN passenger ON pass_id = person_id) NATURAL JOIN promotion
GROUP BY pass_id, type
```

Output:

2 – Kaan Elgin
    “Food”: 35 TL
    “Store”: 40 TL
    “Flight”: 100 TL
5 – Ömer Eren
    “Store”: 10 TL
12 – İrem Ergün
    “Food”: 15 TL
    “Food”: 25 TL
    “Store”: 65 TL
18 – Eren Ekinci
    “Flight”: 50 TL

- A report is used for displaying ticket sales for each passenger with their prices and types as domestic or not.

Query:

```
SELECT pass_id, person_name, price, type AS CASE
            WHEN flight_id IN (SELECT flight_id
                               FROM flight
                               WHERE dep_country = arr_country) THEN “Domestic”
             ELSE “International”
             END
FROM ticket NATURAL JOIN (passenger JOIN person ON pass_id = person_id)
GROUP BY pass_id
```

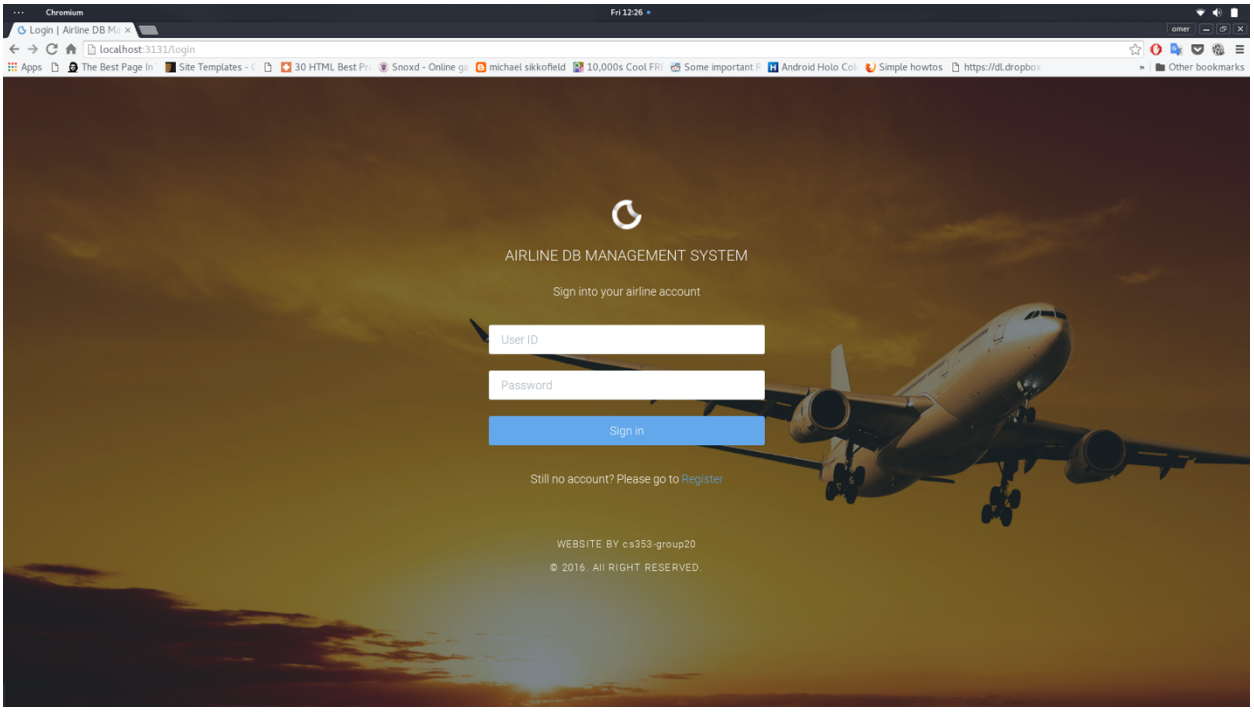
Output:

2 – Kaan Elgin
    “Domestic”: 45 TL
    “International”: 50 TL
5 – Ömer Eren
    “Domestic”: 20 TL
    “Domestic”: 35 TL
    “Domestic”: 40 TL
12 – İrem Ergün
    “International”: 100 TL
18 – Eren Ekinci
    “International”: 150 TL
    “International”: 170 TL

6 User's Manual

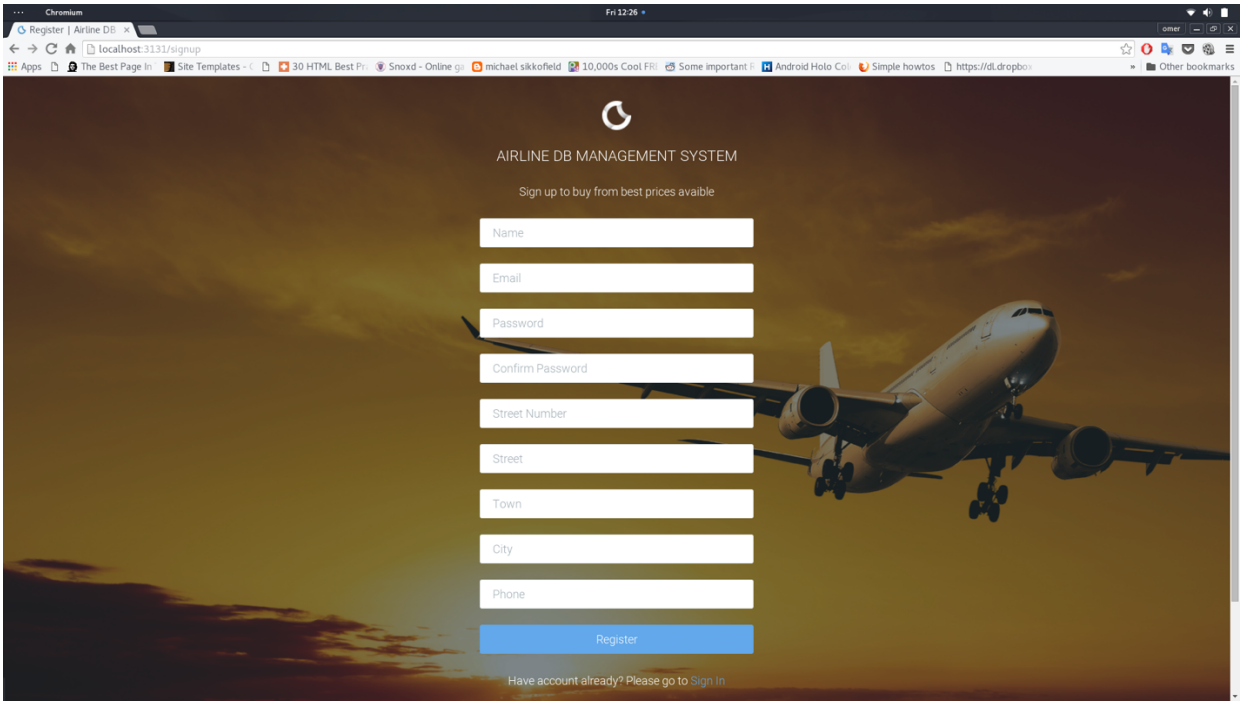
In this section, user manuals for users with different accesses to the system are given. However, Login, Profile, and Signup pages are common for all users. So, they will be described first.

6.1 Login Page



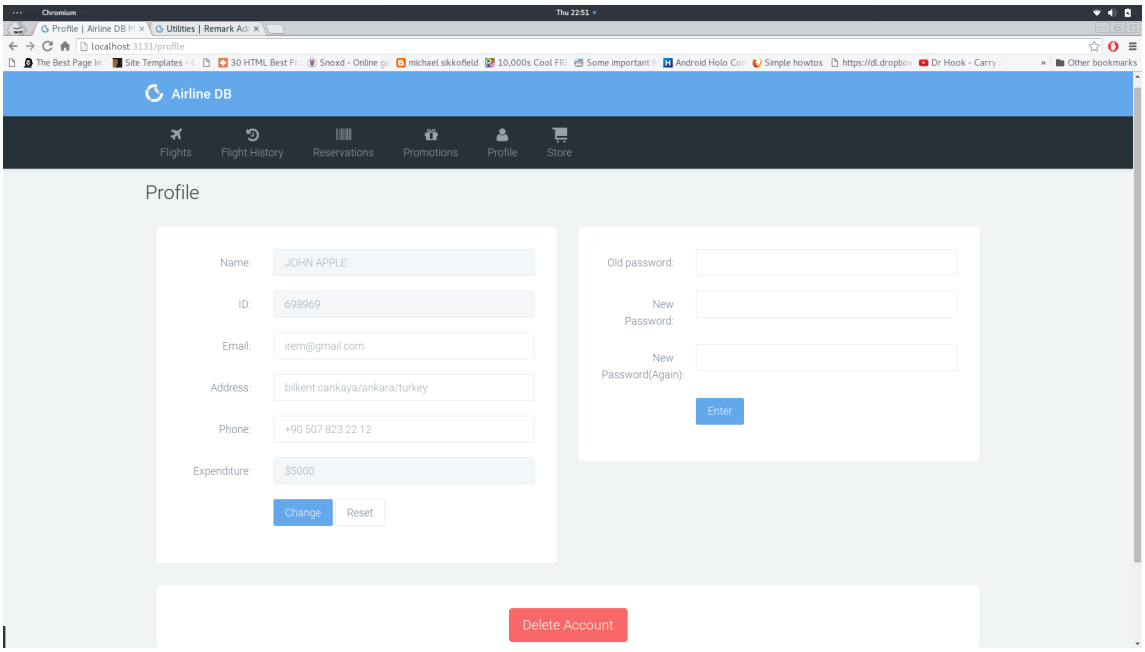
The login page is the first page the user sees when he/she enters the system. If users try to access a page without logging in, they will be redirected to this page, which consists of a form requesting the id and the password information of the users. For people who have not signed up yet, there is also a link directing the user to the Signup page. If the users do not input the login information correctly, they will be faced with an error message and be asked to input the login information again. Every different type of user is directed to a different page after he/she has logged in.

6.2 Signup Page



This page is used for creating accounts to access the system. There are some fields in this page requesting information from the user. After filling in the required information, the users are assigned as passengers or staff by the admin and can use the system by inputting their login information on the Login page.

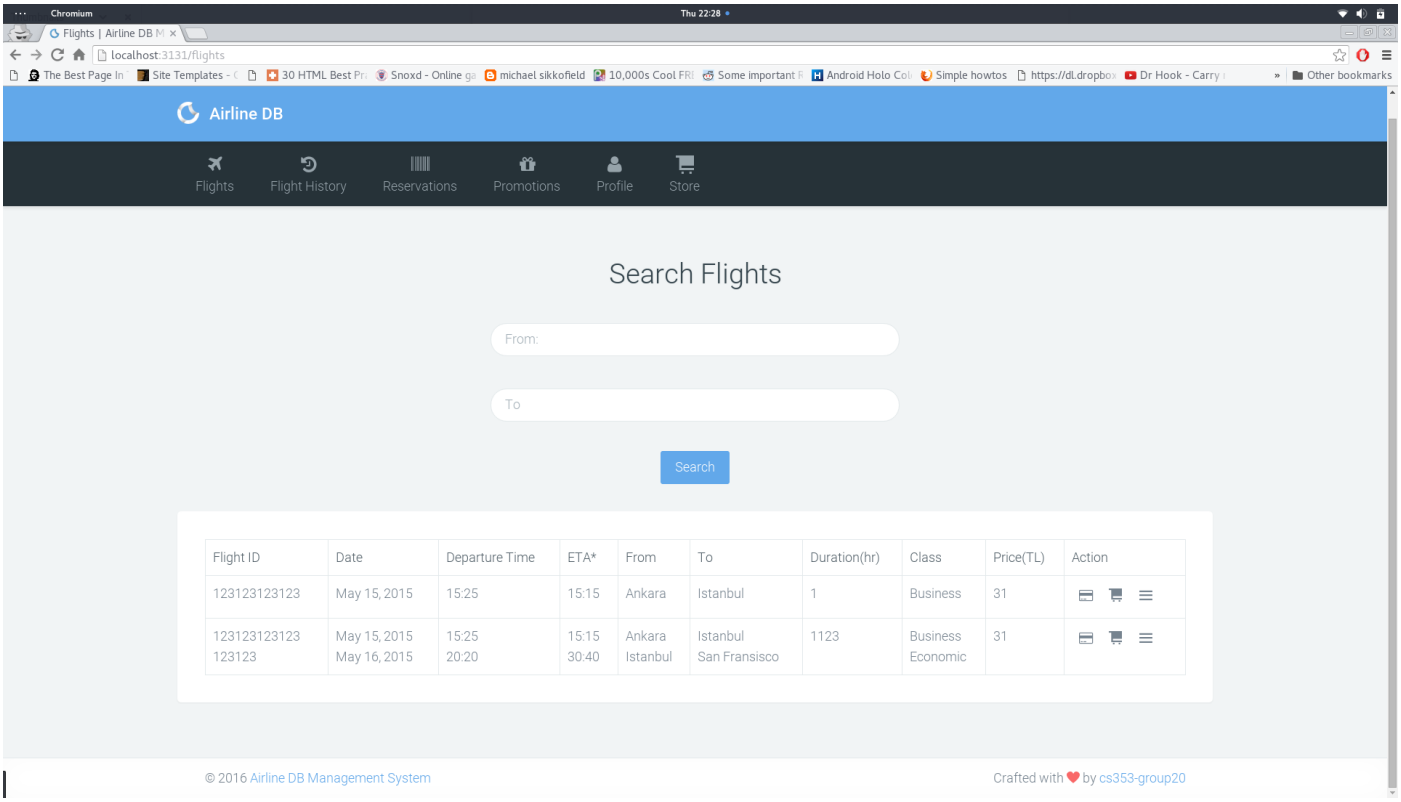
6.3 Profile Page



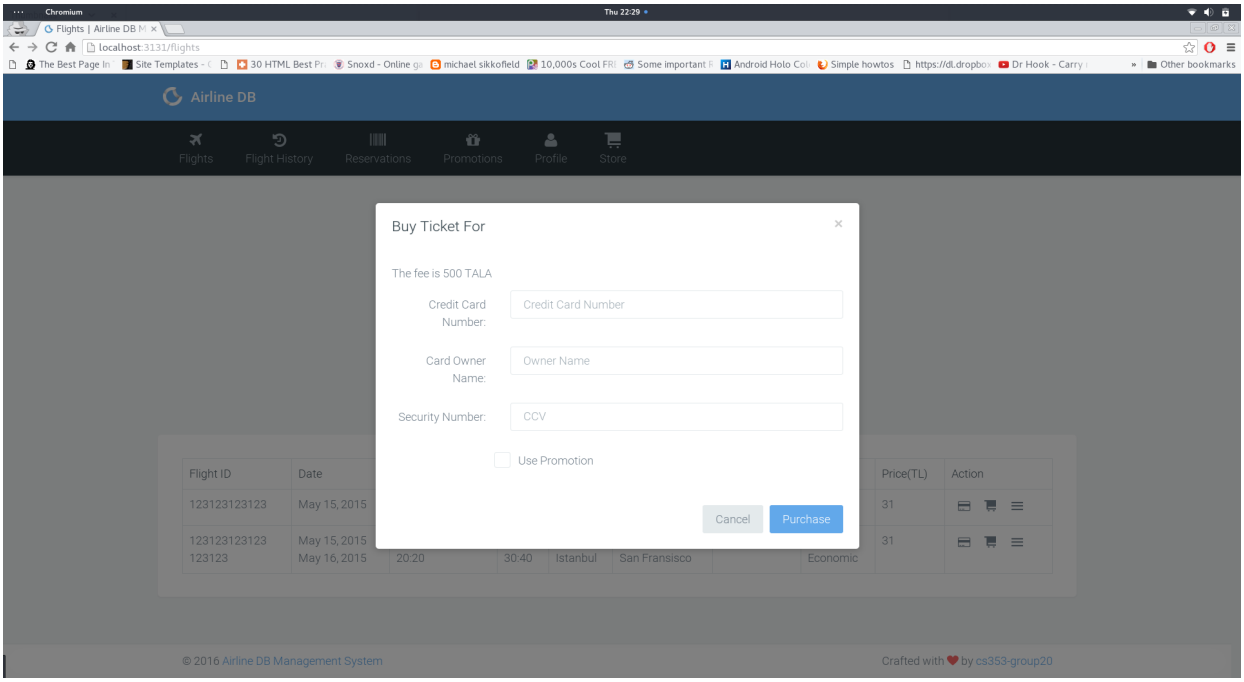
The users can view all of their personal and system related information from this page. They have the chance to update their personal information and change their passwords via this page. Also, they can delete their accounts from via clicking the respective button on this screen.

6.4 User Manual for Customers

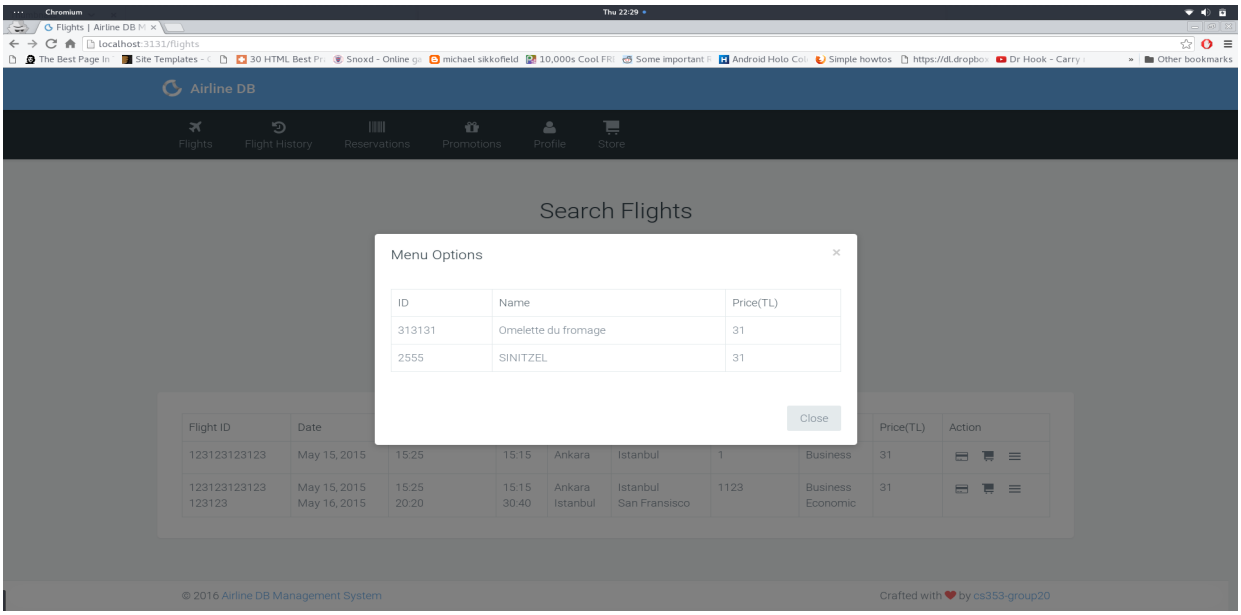
6.4.1 Flights Page



When customers log in to the system, they will first see this page. In this page, the customers are able to browse flights. They can select a path from a location to another location and browse the flights on that destination. The system offers two options for a flight from a place to another, which are direct flight and connecting flight. When there is a direct flight, the system shows a single flight going from the current location to the desired location in one row. When there is a connecting flight, multiple flights are shown in one flight row as seen in the figure above.

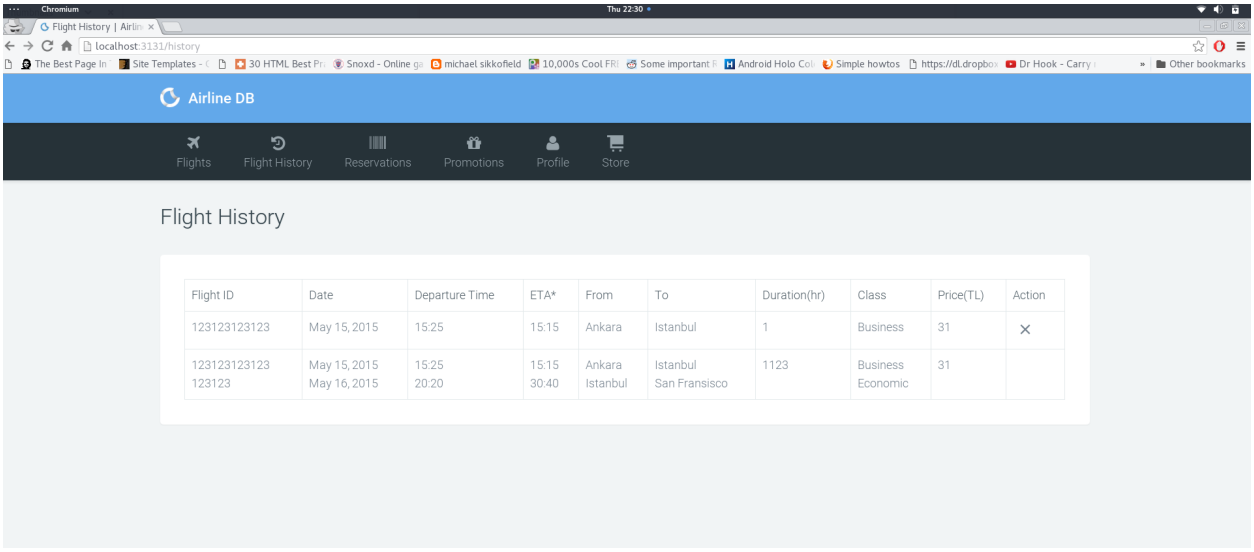


After selecting the flight they desire, the customers can buy a ticket or make a reservation for the selected flight by clicking the icons in that flight row. For making a reservation, the customer needs to specify which seat he/she wants to reserve via a popup screen. While buying a ticket, the customers are shown the fee they need to pay and are asked to input the payment information in a popup screen. They have the option to use flight promotions while making the payment if they have any promotion available.

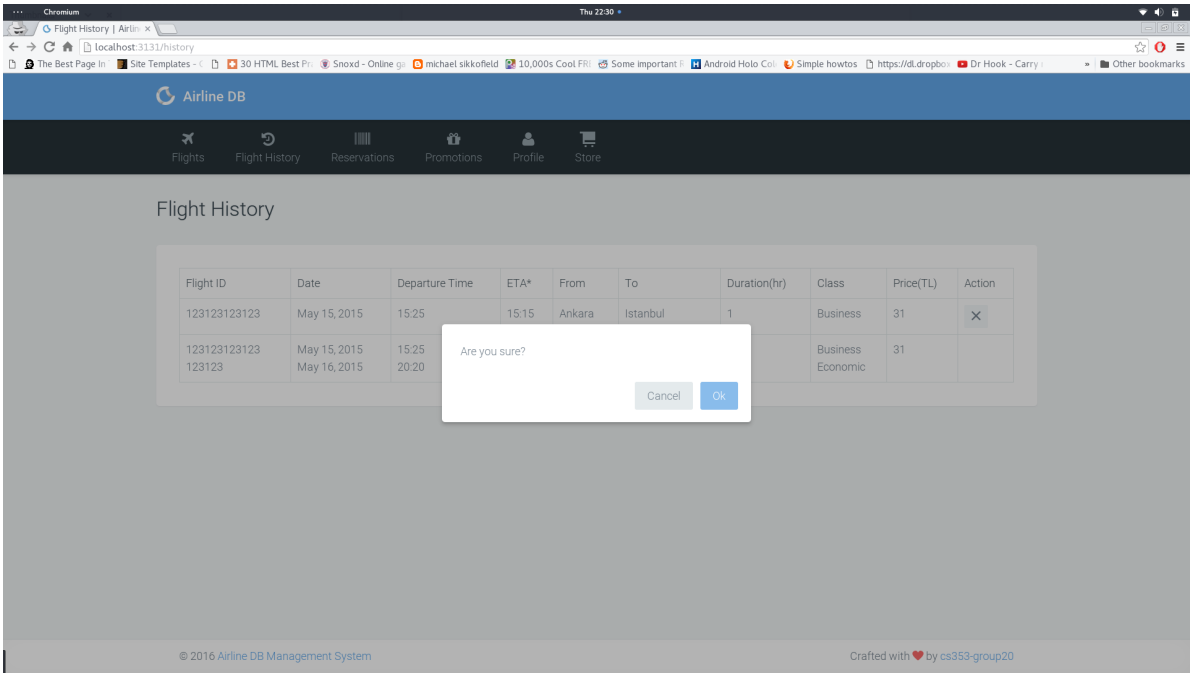


In addition, customers can view the menu options of a flight by clicking the respective icon in the row. In this popup window, the food options are given with respective prices.

6.4.2 Flight History Page

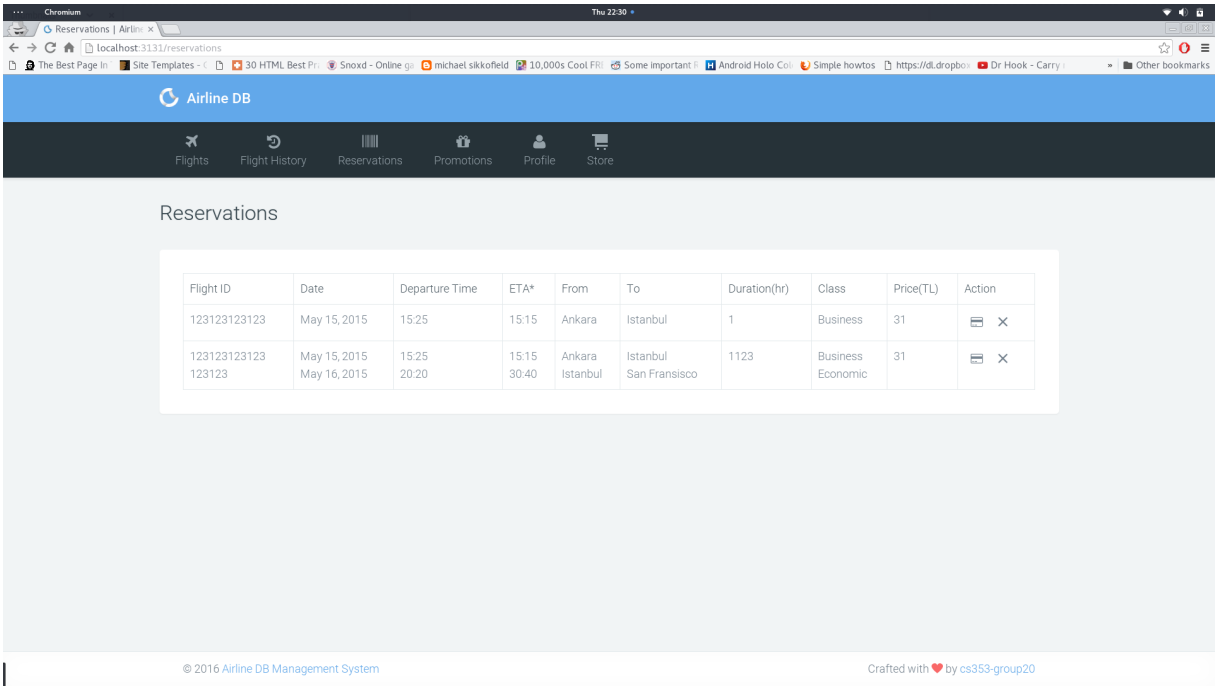


In this page, the customers are able to browse their flight histories. This page will also include the flights that have not happened yet. The customers can cancel those flights from this screen by clicking on the cancel button in the respective row.



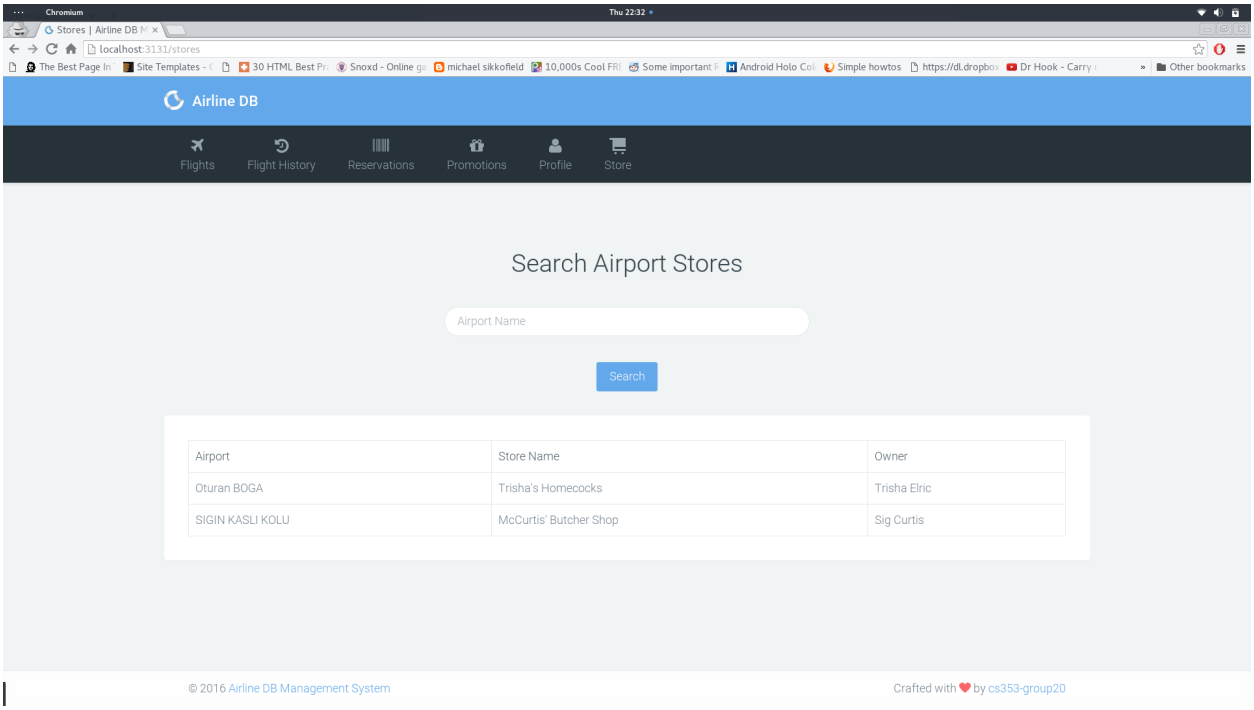
The customers are then asked if they are sure to cancel and are also charged with a fee if they have passed a specified deadline to cancel their flights. Then, they are asked to pay this fee through this screen via a popup window.

6.4.3 Reservations Page



The customers can browse their reservations from this page. They can cancel their reservations by clicking on the button in the respective row. They are asked whether they are sure or not after a cancellation attempt. They can also buy the ticket they have reserved via this screen. If they choose to buy, then they are asked input the payment information in a popup window. They have the option to use a flight promotion for making the payment.

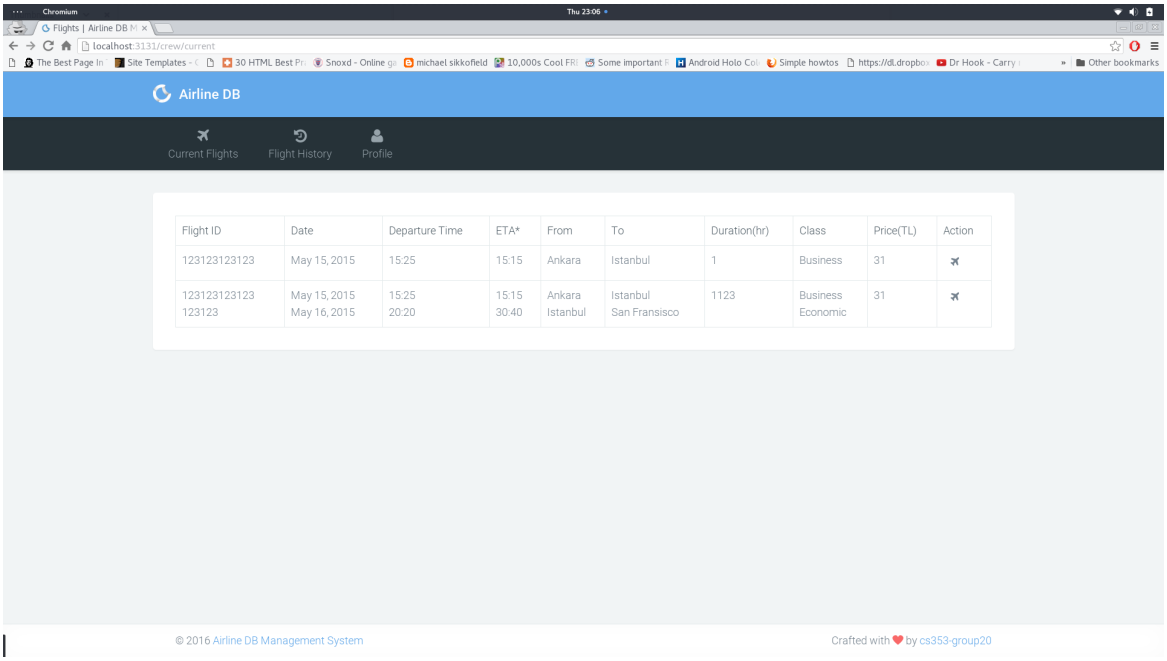
6.4.4 Stores Page



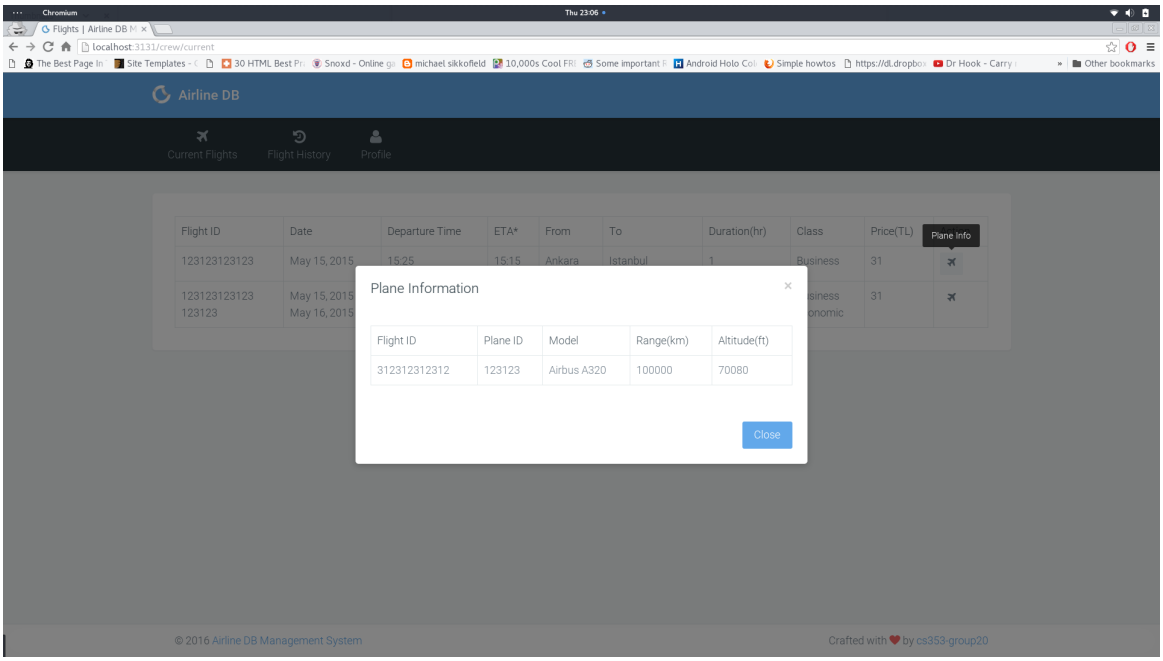
The customers can view the stores belonging to specific airports in this page. They can search for a specific airport and find the stores in that airport. This way, it will be easier for customers to find an appropriate store to spend their store promotions.

6.5 User Manual for Flight Crew

6.5.1 Current Flights Page

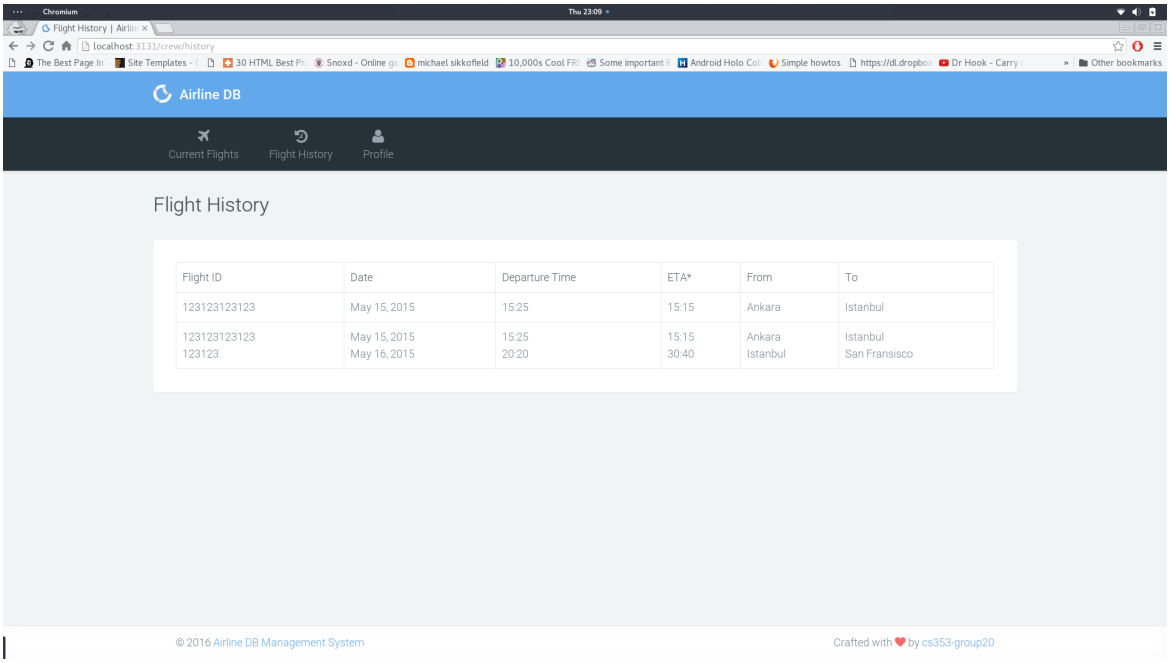


Flight attendants and pilots can view their upcoming flight schedule via this page. They are able to the time and destination of the future flights they are assigned to.



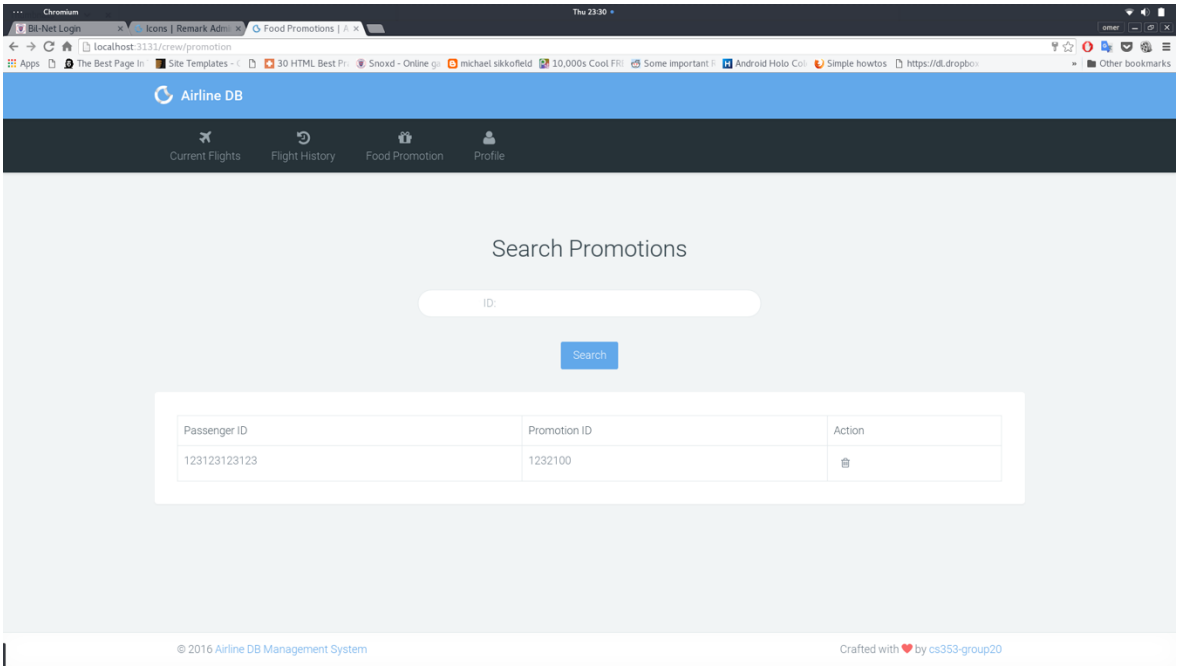
In addition to this, only pilots can view the plane information of a flight via this screen. To do that, they click the plane icon in a specific row, and then, a popup screen comes up giving information about the plane the pilot is going to fly.

6.5.2 Flight History Page



Flight personnel can view their flight histories via this page.

6.5.3 Food Promotions Page

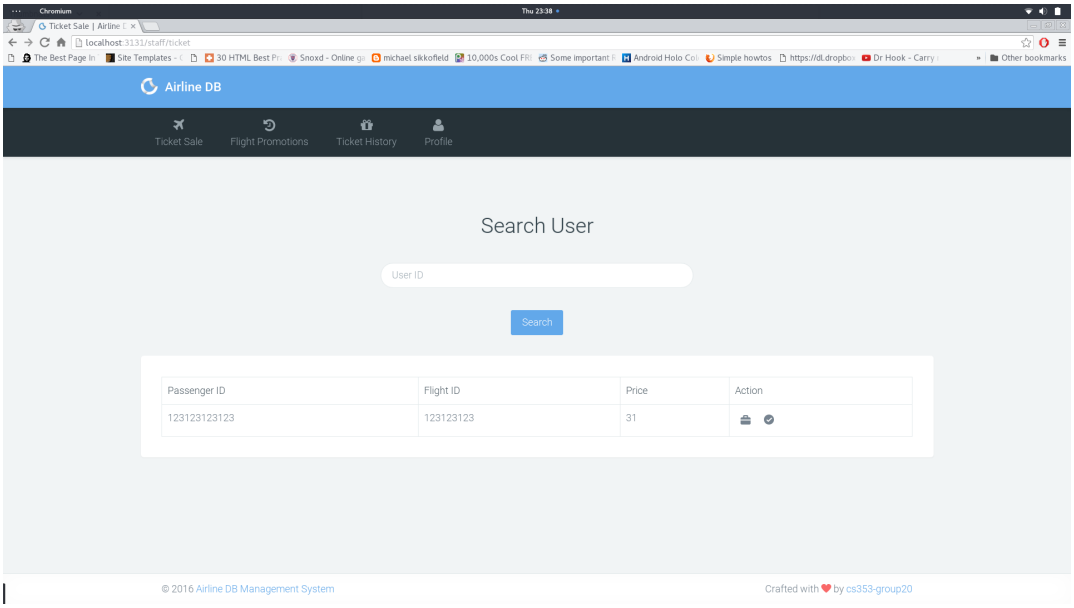


This page is available only for flight attendants. They are responsible for keeping track of the food promotion of the customers on the plane and for recording the usage of food promotion by customers

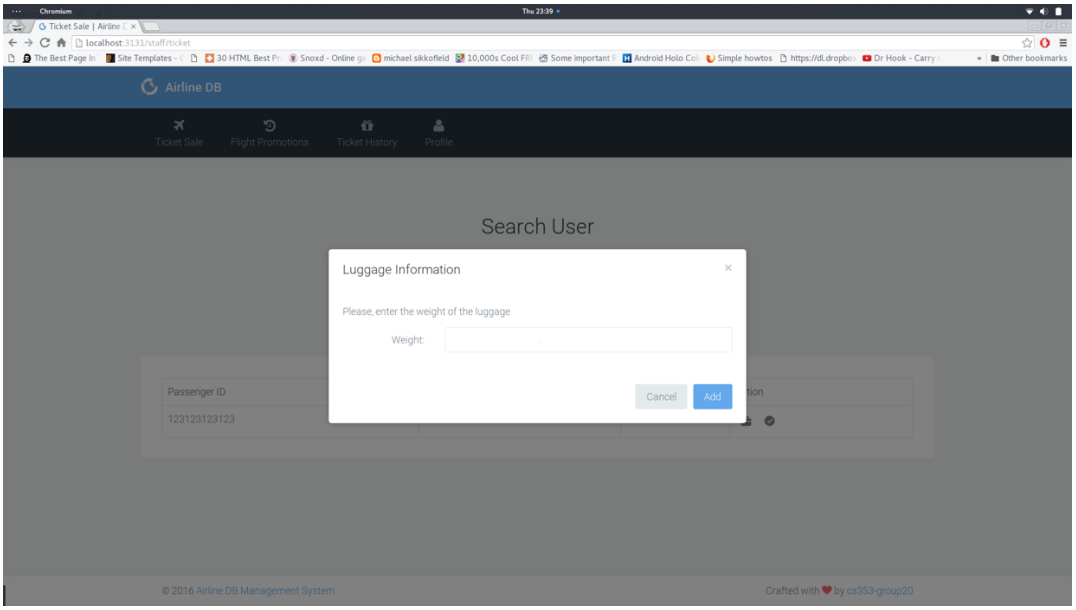
during flights. To achieve that, they search the respective passengers by their id's and delete the respective promotion from the available food promotions list in this screen.

6.6 User Manual for Ticket Staff

6.6.1 Ticket Sale Page



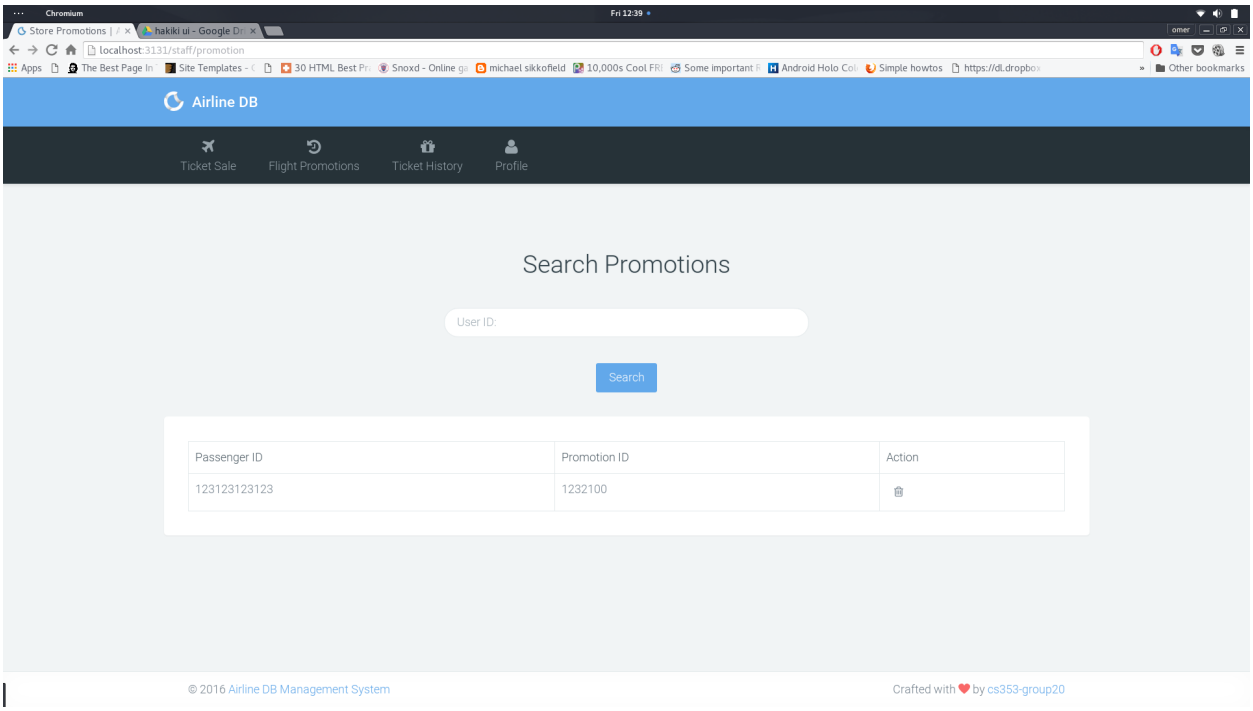
This page is for ticket staff who are selling a ticket to a customer face to face in the airport. The customers are searched via their id's and they buy the tickets they want with the courtesy of the ticket staff.



Staff also enters the luggage information of customers (in a popup window which is reachable from clicking luggage icon in the respective row) and demands an extra payment if their luggage exceeds the

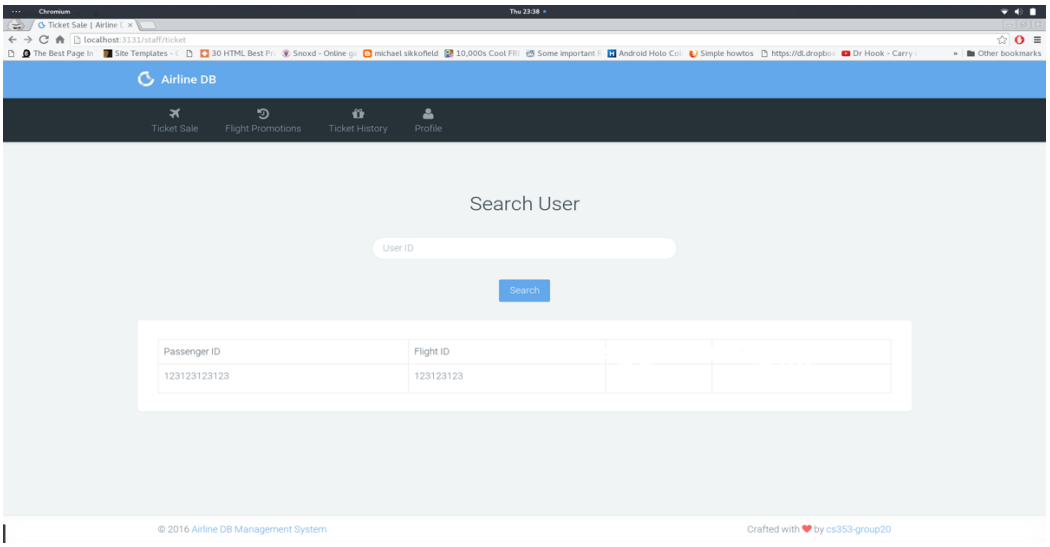
standard weight limit. Then, the staff takes the respective amount of money from customers and completes the sale. Customers can also pay with their flight promotions if they have any available.

6.6.2 Flight Promotions Page



If customers choose to pay with their promotions, ticket staff records usage and deletes the used promotion from available flight promotion list. To find the respective user, ticket staff searches with the id of that customer.

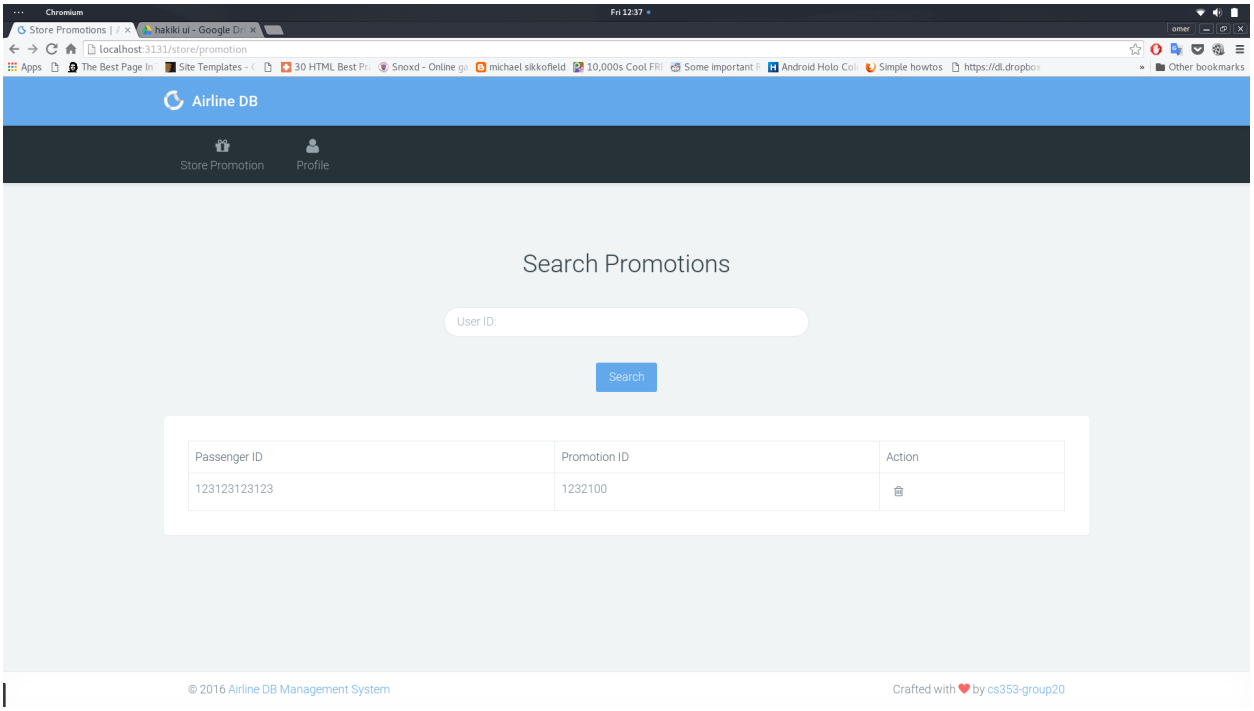
6.6.3 Ticket History Page



Ticket staff can view the ticket history of a customer via this screen. The ticket history of each customer are kept separately. In order to view the ticket history of a specific customer, staff needs to do a search with the id of that customer.

6.7 User Manual for Store Staff

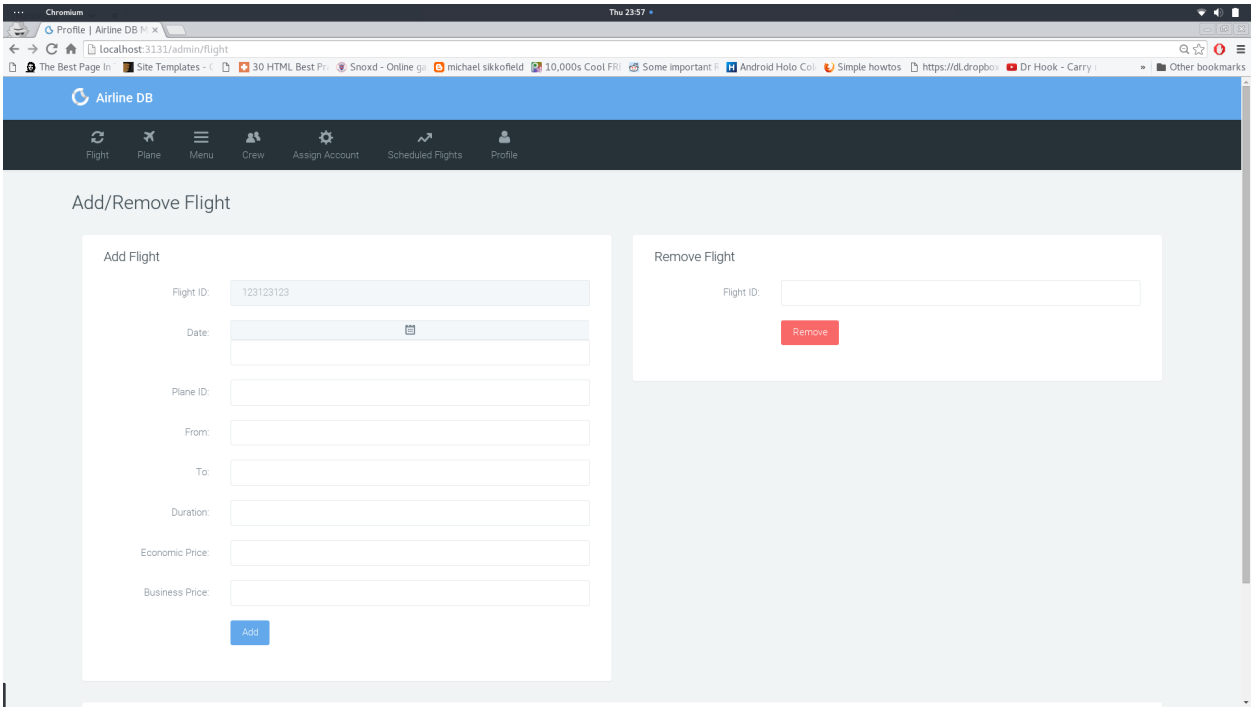
6.7.1 Store Promotions Page



Store staff are responsible for keeping track of the usage of store promotions. When a store promotion is used by a customer, store staff makes a search with the id of that customer and deletes the respective promotion (by clicking the ‘Delete’ button) from the available store promotions list via this page.

6.8 User Manual for Admin

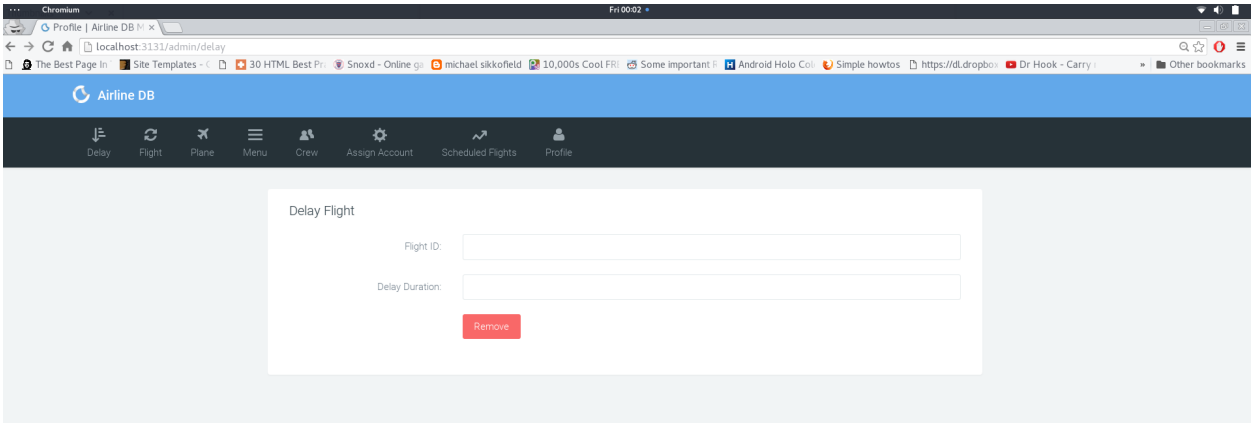
6.8.1 Add/Remove Page(s)



The admin of the system has the ability to add or remove flights. He needs to input the required information about a flight and then press ‘Add’ button to add a flight. To remove a flight, he only needs to enter the id of that flight and click ‘Remove’ button.

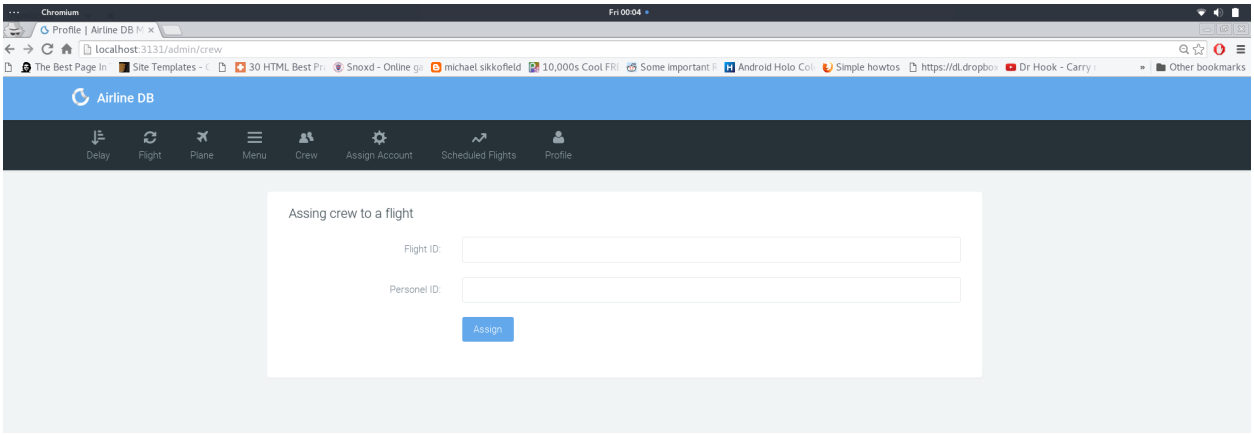
Add/Remove Plane, Add/Remove Food Menu Option, Add/Remove Store pages are very similar to Add/Remove Flight Page. For Add/Remove Plane screen, the information the admin needs to input consists of model, capacity, range and altitude of the respective plane. For Add/Remove Food Option, he needs to input id of the flight that the promotion belongs to, the price of the promotion and the name of the promotion. For Add/Remove Store screen, he needs to enter the airport name, city that the respective airport is in and the country, the name of the store and the owner of the store. Then, he needs to press ‘Add’ button to add. Again, to delete an entity from any of the pages, he needs to enter the id of the entity to be deleted.

6.8.2 Delay Flight Screen



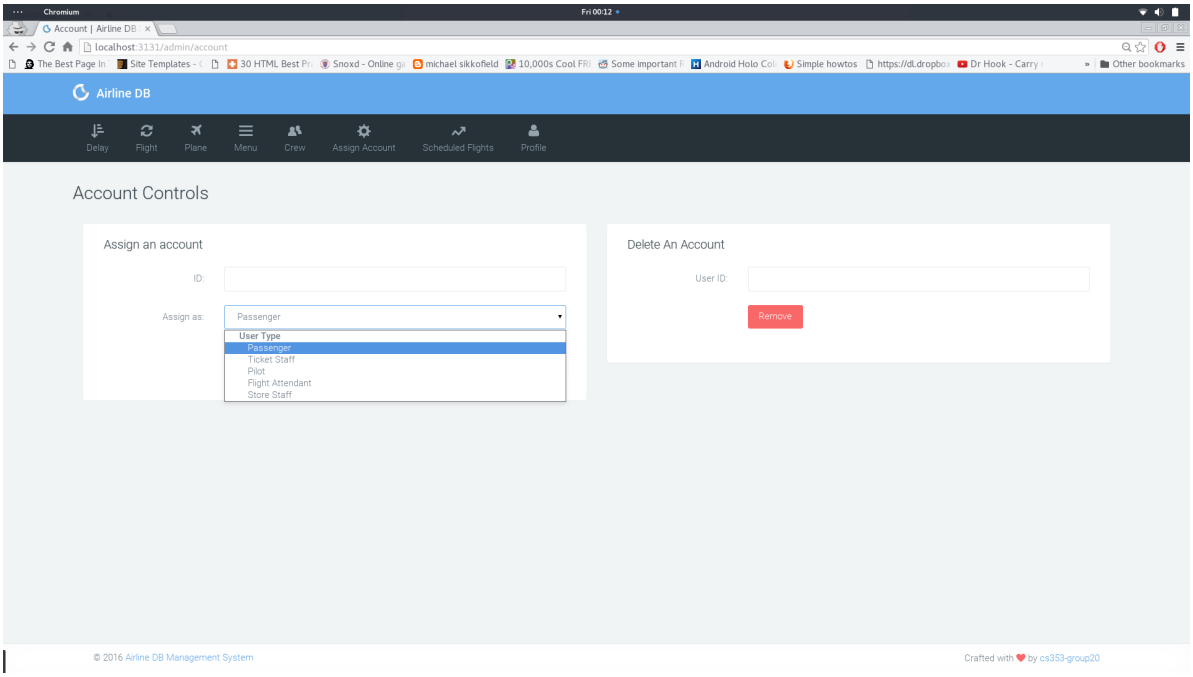
The admin is responsible for inputting the delay of a flight. To do that, he needs to enter the id of the flight which has been delayed and the delay duration of that flight to the system via this screen. After this operation, all the flight information shown to the other users are altered.

6.8.3 Assign Flight Personnel Page



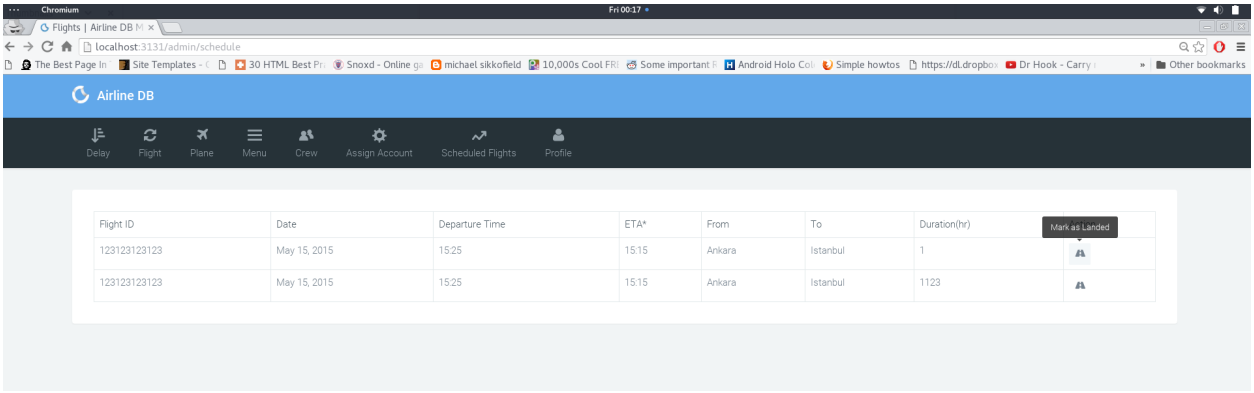
The admin assigns flight personnel to their respective flights through this page. To do that, he inputs the id of the flight personnel to be assigned, and the id of the flight he/she will be assigned to.

6.8.4 Assign/Delete Account Page



When users sign up to the system, admin is required to assign them according to their user types. For example, when a user creates an account and is a customer, the admin enters the id of that user and selects the 'Customer' area from the 'Assign As' scroll bar. This action is performed from this screen. Also, using this screen, the admin can delete the account of any user by inputting the id of that user.

6.8.5 Scheduled Flights Page



Using this page, the admin can view the upcoming flights. When a plane lands and the flight has ended, the admin marks that flight as landed by clicking on the 'road' icon in the respective row and that flight is removed from this screen.